Windows Kernel Reference Count Vulnerabilities - Case Study

Mateusz "j00ru" Jurczyk

@ ZeroNights E.0x02
November 2012

```
PS C:\Users\j00ru> whoami

        nt authority\system
```

- Microsoft Windows internals fanboy
- Also into reverse engineering and low-level software security
- Currently in Switzerland working at Google

Why this talk?

- Lost of stuff in a sandbox
  - Google Chrome, Adobe Reader, Apple Safari, pepper plugins, ...
  - Escapes are becoming valuable
- Also, escapes are super exciting!
  - https://krebsonsecurity.com/2012/11/experts-warn-of-zero-day-exploit-for-adobe-reader/ (just recently)
  - ... really, is this so shocking?
- "New" old class of bugs in the Windows kernel
- Otherwise, a bunch of technically interesting bugs

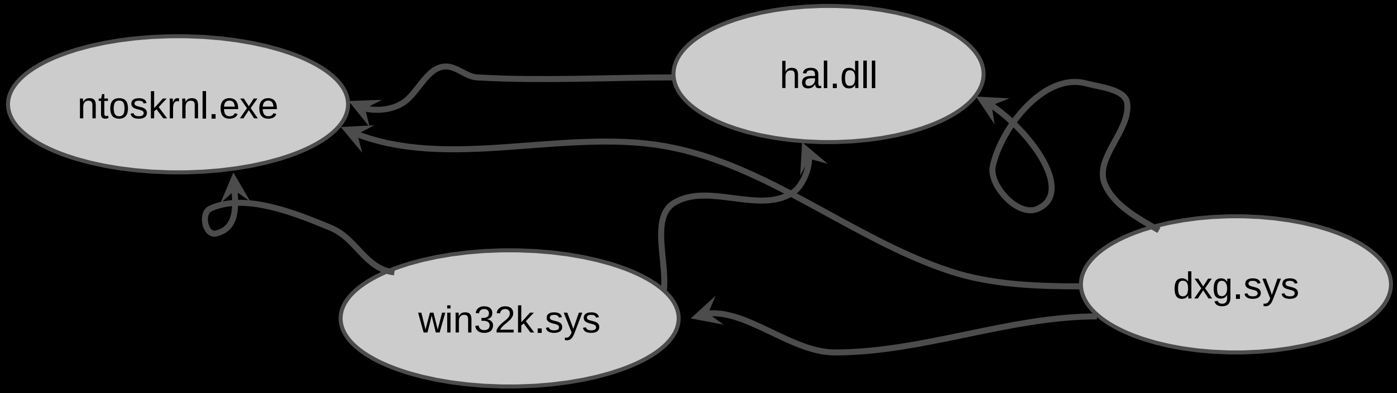
Topics covered

- Reference counting philosophy and problems
- Case study
  a. 1-day (NT Object Manager *PointerCount* weakness)
  b. 0-day (generic device driver image use-after-free)
  c. CVE-2010-2549 (win32k!NtUserCheckAccessForIntegrityLevel use-after-free)
  d. CVE-2012-2527 (win32k!NtUserAttachThreadInput use-after-free)
  e. CVE-2012-1867 (win32k!NtGdiAddFontResource use-after-free)
- Mitigations and lessons learned

Reference counting

Fundamentals

- From now on, considering ring-0 refcounting
- System state → graph
    - resources → nodes
    - dependencies (refs) → directed edges
    - lonely node → destroy
        - dynamic memory management = vulnerabilities



Fundamentals

- In the graph scenario, a vertex doesn't have to know who points at him
  - Just the total number
- Common expression in garbage collectors:

```
if (!pObject->Refcount) {
  free(pObject);
}
```

- Unsurprisingly, refcounting is usually implemented using plain integers

Fundamentals

- Typical code pattern

`pObject` guaranteed to persist

```
POBJECT pObject = TargetObject;
PCLIENT pClient = ClientObject;

pObject->Refcount++;
pClient->InternalPtr = pObject;

/* Perform operations on pClient assuming
initialized InternalPtr */

pClient->InternalPtr = NULL;
pObject->Refcount--;
```

Fundamentals

- Windows kernel primarily written in C
- Everything is (described by) a structure
- Lack of common interface to manage references
  - Implemented from scratch every single time when needed...
  - ... always in a different way

Examples?

```
kd> dt _OBJECT_HEADER
nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   +0x008 HandleCount      : Int8B
   +0x008 NextToFree       : Ptr64 Void
   +0x010 Lock             : _EX_PUSH_LOCK
[...]
```

```
kd> dt _LDR_DATA_TABLE_ENTRY
nt!_LDR_DATA_TABLE_ENTRY
[...]
   +0x068 Flags            : Uint4B
   +0x06c LoadCount        : Uint2B
   +0x06e TlsIndex         : Uint2B
   +0x070 HashLinks        : _LIST_ENTRY
[...]
```

```
kd> dt tagQ
win32k!tagQ
   +0x000 mlInput          : tagMLIST
[...]
   +0x070 hwndDblClk       : Ptr64 HWND__
   +0x078 ptDblClk         : tagPOINT
   +0x080 ptMouseMove      : tagPOINT
   +0x088 afKeyRecentDown  : [32] UChar
   +0x0a8 afKeyState       : [64] UChar
   +0x0e8 caret            : tagCARET
   +0x130 spcurCurrent     : Ptr64 tagCURSOR
   +0x138 iCursorLevel     : Int4B
   +0x13c QF_flags         : Uint4B
   +0x140 cThreads         : Uint2B
   +0x142 cLockCount       : Uint2B
[...]
```

Reference counting: problems

Logical issues

- Crucial requirement: refcount must be adequate to number of references by pointer
- Obviously, two erroneous conditions
  - Refcount is inadequately small
  - Refcount is inadequately large
- Depending on the context, both may have serious implications

Overly small refcounts

- Two typical reasons
  - Reference-by-pointer without refcount incrementation
  - More decrementations in a *destroy* phase than incrementations performed before
- Foundation of modern user-mode vulnerability hunting (web browsers et al)
  - http://zerodayinitiative.com/advisories/published/
  - http://blog.chromium.org/2012/06/tale-of-two-pwnies-part-2.html
  - https://www.google.pl/#q=metasploit+use-after-free
  - ...

Overly small refcounts

- Typical outcome in ring-3

object vtable lookup + call

```
mov eax, dword ptr [ecx]
mov edx, dword ptr [eax+70h]
call edx
```

- Still use-after-free in ring-0, but not so trivial
  - almost no vtable calls in kernel
  - exploitation of each case is bug specific and usually requires a lot of work
  - kernel pools feng shui is far less developed and documented compared to userland
  - Tarjei Mandt has exploited a few, check his BH slides and white-paper

Overly large refcounts

- Expected result → resource is never freed
  - Memory leak
  - Potential DoS via memory exhaustion
  - Not very useful
- But refcounts are integers, remember?
  - Finite precision.
  - Integer arithmetic problems apply!
  - Yes, we can try to overflow
- This can become a typical "small refcount" problem
  - use-after-free again

Reference count leaks

- If we can trigger a leak for free, it's exploitable

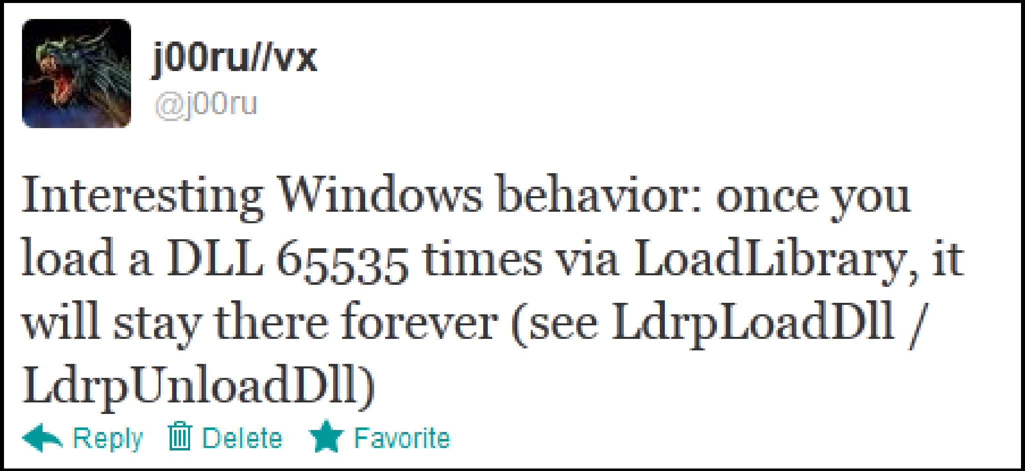
```
while (1) {
  TriggerRefcountLeak(pObject);
}
```

- Unless the integer range is too large
  - uint16_t is not enough
  - uint32_t is (usually) not enough anymore
  - uint64_t is enough

Reference count leaks

- Or unless object pinning implemented (`ntdll!LdrpUpdateLoadCount2`)

```
if (Entry->LoadCount != 0xffff) {
  // Increment or decrement the refcount
}
```



j00ru//vx
@j00ru

Interesting Windows behavior: once you load a DLL 65535 times via LoadLibrary, it will stay there forever (see LdrpLoadDll / LdrpUnloadDll)

Legitimately large refcounts

- Sometimes even those can be a problem
- We can bump up refcounts up to a specific value
- Depends on bound memory allocations

never happens

Per-iteration byte limit	Reference counter size
impossible	64 bits
0-2 bytes	32 bits
16,384 - 131,072 bytes	16 bits
4,194,304 - 33,554,432 bytes	8 bits

Perfect reference counting

Qualities

- Implementation: 32-bit or 64-bit (safe choice) integers.
- Implementation: sanity checking, e.g. `refcount ≥ 0x80000000 ⇒ bail out`
- Usage: `reference# = dereference#`
    - Random idea: investigate system state at shutdown
- Usage: never use object outside of its reference block
- Mitigation: *reference typing*

Reference counting bugs: case study

NT Object Manager PointerCount weakness

- Manages common resources
  - files, security tokens, events, mutants, timers, ...
  - around 50 types in total (most very obscure)
- Provides means to (de)reference objects
  - Public kernel API functions
    - ObReferenceObject, ObReferenceObjectByHandle, ObReferenceObjectByHandleWithTag, ObReferenceObjectByPointer, ObReferenceObjectByPointerWithTag, ObReferenceObjectWithTag
    - ObDereferenceObject, ObDereferenceObjectDeferDelete, ObDereferenceObjectDeferDeleteWithTag, ObDereferenceObjectWithTag
  - Extensively used by the kernel itself and third-party drivers

NT Object Manager PointerCount weakness

Fundamentals

- Each object comprised of a header + body
  - Header common across all objects, body specific to type (e.g ETHREAD, EPROCESS, ERESOURCE)

```
kd> dt _OBJECT_HEADER
win32k!_OBJECT_HEADER
   +0x000 PointerCount      : Int4B
   +0x004 HandleCount       : Int4B
[...]
   +0x008 Type              : Ptr32 _OBJECT_TYPE
[...]
   +0x018 Body              : _QUAD
```

native word-wide reference counters

type specifier

type specific structure

NT Object Manager PointerCount weakness

Fundamentals

- Two reference counters
  - `PointerCount` - # of direct kernel-mode pointer references
  - `HandleCount` - # of indirect references via `HANDLE` (both ring-3 and ring-0)
- Object free condition `(PointerCount == 0) && (HandleCount == 0)`

NT Object Manager PointerCount weakness

- Security responsibility put on the caller
  - Allows arbitrary number of decrementations
  - Allows reference count integer overflows
- Excessive dereferences rather uncommon
  - CVE-2010-2549 is the only I can remember
- Reference leaks on the other hand...
  - can *theoretically* only lead to memory leak
    - who'd care?
  - *sometimes you just forget to close something*
  - much more popular (in third-parties, not Windows)

NT Object Manager PointerCount weakness

- Userland can't overflow `HandleCount`
  - At least 32GB required to store four billion descriptors.
  - `HANDLE` address space is four times smaller than a native word.
- But random drivers can overflow `PointerCount`
  - grep through %system32%\drivers?

```
< Binary file ./cpqdap01.sys matches
< Binary file ./isapnp.sys matches
< Binary file ./modem.sys matches
< Binary file ./nwlnkipx.sys matches
< Binary file ./pcmcia.sys matches
< Binary file ./sdbus.sys matches
< Binary file ./wmilib.sys matches
```

Import a *Reference*, but no *Dereference* symbol.

NT Object Manager PointerCount weakness

- Refcount leaks are as dangerous as double derefs (only on 32-bit platforms)
    - just take longer to exploit
- Had a chat with *Microsoft security*
- A few months later, Windows 8 ships with a fix:

```
[...]
v8 = _InterlockedIncrement((signed __int32 *)v5);
if ( (signed int)v8 <= 1 )
  KeBugCheckEx(0x18u, 0, ObjectBase, 0x10u, v8);
[...]
```

" The REFERENCE_BY_POINTER bug check has a value of 0x00000018. This indicates that the reference count of an object is illegal for the current state of the object. "

NT Object Manager PointerCount weakness

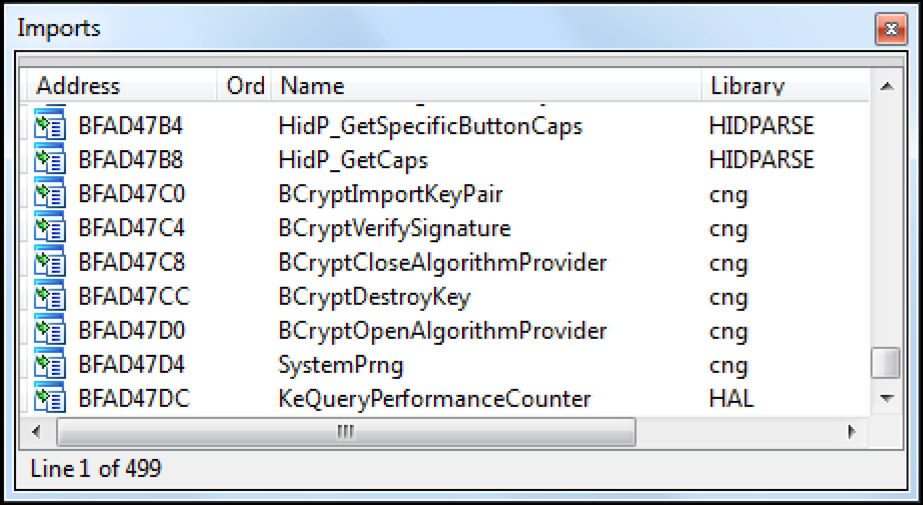
- Ken Johnson and Matt Miller covered this and other mitigations during their BH USA 2012 presentation
  - "Exploit Mitigation Improvements in Windows 8", check it out
- Mitigation only released for Windows 8
  - older platforms still affected
  - go and find your own unpaired *ObReferenceObject* invocations?

Device driver image use-after-free

- Many drivers loaded in Windows at any time

```
kd> lm
start    end         module name
80ba0000 80ba8000   kdcom      (deferred)
8281f000 82c31000   nt         (pdb symbols)
82c31000 82c68000   hal        (deferred)
82e00000 82e25000   CLASSPNP   (deferred)
[...]
```

- They import from each other extensively


Imports

Address	Ord	Name	Library
BFAD47B4		HidP_GetSpecificButtonCaps	HIDPARSE
BFAD47B8		HidP_GetCaps	HIDPARSE
BFAD47C0		BCryptImportKeyPair	cng
BFAD47C4		BCryptVerifySignature	cng
BFAD47C8		BCryptCloseAlgorithmProvider	cng
BFAD47CC		BCryptDestroyKey	cng
BFAD47D0		BCryptOpenAlgorithmProvider	cng
BFAD47D4		SystemPrng	cng
BFAD47DC		KeQueryPerformanceCounter	HAL

Line 1 of 499

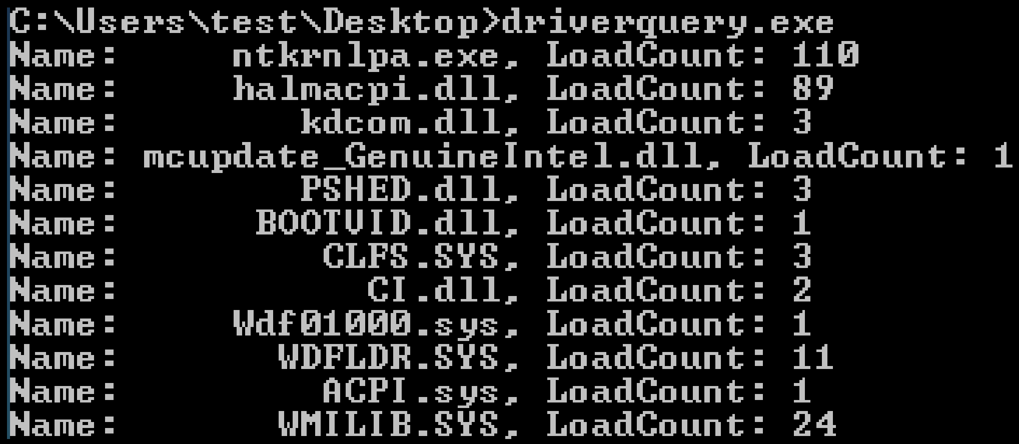
Device driver image use-after-free

- In other words, drivers are resources that reference each other
  - refcounts!
- Each described by `LDR_DATA_TABLE_ENTRY`

```
kd> dt _LDR_DATA_TABLE_ENTRY
nt!_LDR_DATA_TABLE_ENTRY
   [...]
   +0x024 FullDllName      : _UNICODE_STRING
   +0x02c BaseDllName      : _UNICODE_STRING
   +0x034 Flags            : Uint4B
   +0x038 LoadCount        : Uint2B
   +0x03a TlsIndex         : Uint2B
```

16-bit only!

Device driver image use-after-free



```
C:\Users\test\Desktop>driverquery.exe
Name:     ntkrnlpa.exe, LoadCount: 110
Name:     halmacpi.dll, LoadCount: 89
Name:        kdcom.dll, LoadCount: 3
Name: mcupdate_GenuineIntel.dll, LoadCount: 1
Name:        PSHED.dll, LoadCount: 3
Name:      BOOTVID.dll, LoadCount: 1
Name:         CLFS.SYS, LoadCount: 3
Name:           CI.dll, LoadCount: 2
Name:     Wdf01000.sys, LoadCount: 1
Name:       WDFLDR.SYS, LoadCount: 11
Name:         ACPI.sys, LoadCount: 1
Name:       WMILIB.SYS, LoadCount: 24
```

- If we load a driver that imports from e.g. fwpkclnt.sys 65,536 times, *LoadCount* is overflown.
  - must be a different path every time.
- Smallest default drivers take up 8kB - 65kB of virtual address space.
  - still within reasonable limits on X86-64 (within 4GB)

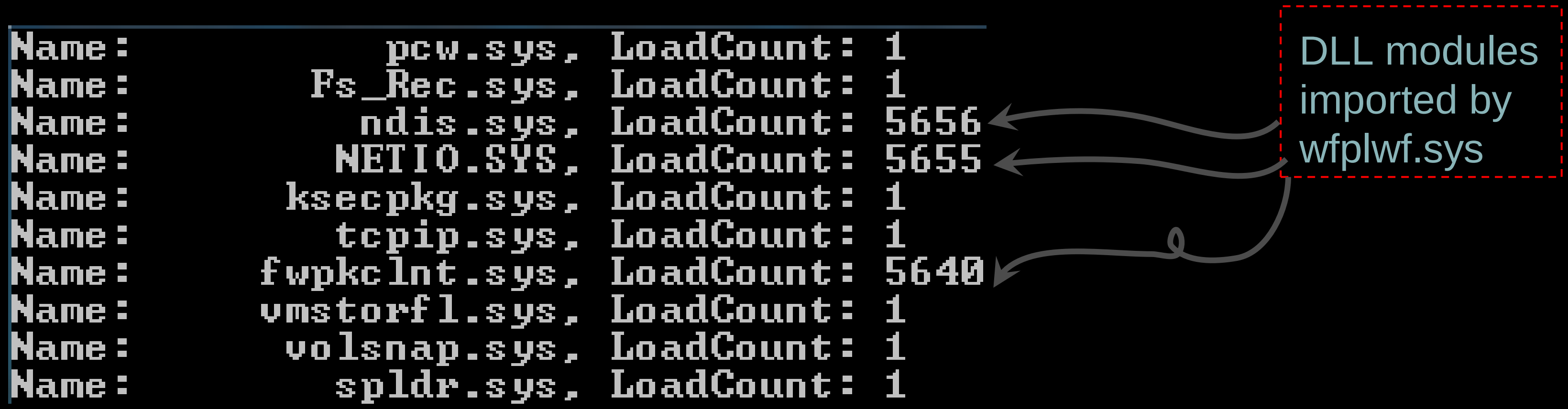
Device driver image use-after-free

- Not all drivers can be unloaded, even for refcount=0
  - there's a concept of kernel DLLs
    - not stand-alone, only loaded as dependencies
    - can be recognized by DllInitialize / DllUnload exports
    - examples: usbport.sys, msrpc.sys, Classpnp.sys
- Exploitation plan:
  - Find a small driver importing from a kernel DLL to load multiple times
  - Find another such driver which fails to load.
  - Overflow DLL refcount using driver A, then free using driver B.

Device driver image use-after-free

- Exemplary setting: use wfplwf.sys to overflow the netio.sys (DLL) refcount
- Use tcpip.sys to trigger the free(netio.sys)
- Works good!

Refcounts in the middle of an attack:



```
Name:          pcw.sys, LoadCount: 1
Name:       Fs_Rec.sys, LoadCount: 1
Name:         ndis.sys, LoadCount: 5656
Name:        NETIO.SYS, LoadCount: 5655
Name:      ksecpkg.sys, LoadCount: 1
Name:        tcpip.sys, LoadCount: 1
Name:     fwpkclnt.sys, LoadCount: 5640
Name:    vmstorfl.sys, LoadCount: 1
Name:      volsnap.sys, LoadCount: 1
Name:        spldr.sys, LoadCount: 1
```

Device driver image use-after-free

Effective result

```
[…]
<Unloaded_NETIO.SYS>+0x1b70:
88557b70 ??              ???
Resetting default scope
[…]
 0: kd> kb
ChildEBP RetAddr  Args to Child
8078a654 […] <Unloaded_NETIO.SYS>+0x1b70
8078a668 […] tcpip!CheckInboundBypass+0x1f
8078a810 […] tcpip!WfpAleFastUdpInspection+0x55
[…]
```

Device driver image use-after-free

- Impact
  - Administrative rights required
  - Therefore, only admin → ring-0 privilege escalation
  - Useful for subverting *Driver Signature Enforcement*
    - not much else

Device driver image use-after-free

Metrics

- Memory
  - wfplwf.sys takes 0x7000 bytes (28kB) of virtual memory.
  - 0x10000 (65,536) instances = ~2GB total.
- CPU time
  - Platform: Windows 7 64-bit, 4-core VMware Player, Intel i7-3930K @ 3.20GHz
  - ~100 loads per second.
  - 65,536 loads ~ 655 seconds ~ 10 minutes

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- On Wed, 30 Jun 2010 *Microsoft-Spurned Researcher Collective* dropped a 0-day at full disclosure.
    - Windows Vista / 2008 only
    - included a link to j00ru.vexillium.org :-/
- Turned out to be a trivial double-deref when accessing a *PsProcessType* object
    - Managed by the NT Object Manager

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

Faulty call chain

- `win32k!NtUserCheckAccessForIntegrityLevel`
    - `win32k!LockProcessByClientId`
        - `win32k!LockProcessByClientIdEx`
            - `nt!PsLookupProcessByProcessId`
                - `nt!ObReferenceObjectSafe`
            - `nt!PsGetProcessSessionId`
        - `nt!ObfDereferenceObject`
    - `nt!ObfDereferenceObject`

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- Referenced once

`nt!PsLookupProcessByProcessId`

```
PAGE:006167A9  call    @ObReferenceObjectSafe@4
```

- Dereferenced twice

`win32k!LockProcessByClientId`

```
.text:BF88E63B  call    ds:__imp_@ObfDereferenceObject@4
```

`win32k!NtUserCheckAccessForIntegrityLevel`

```
.text:BF92D329  call    ds:__imp_@ObfDereferenceObject@4
```

- Broke the `reference# = dereference#` rule

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- Bug allows arbitrary decrementation of `PointerCount` of an object.

- Conditions
  - Must be a process (`PsProcessType`)
  - In a different terminal session than caller (`process session id != gSessionId`)
    - System, smss.exe, lsass.exe, ...
    - Remote Desktop Services applications

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- Exploitation concept
  a. Find a process with `HandleCount = 0`
  b. Free the object by dropping `PointerCount` to 0
  c. Spray object memory with controlled data.
  d. ???
  e. PROFIT!
- smss.exe looks good

```
PROCESS 8c10b628  SessionId: none  Cid: 0194    Peb: 7ffda000  ParentCid: 0004
    DirBase: 0015c020  ObjectTable: 87fc6fc8  HandleCount:  28.
    Image: smss.exe

kd> !object 8c10b628
Object: 8c10b628  Type: (8465aec0) Process
    ObjectHeader: 8c10b610 (old version)
    HandleCount: 0  PointerCount: 22
```

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

Crash easy to trigger

```
TRAP_FRAME:  90706b0c -- (.trap 0xffffffff90706b0c)
ErrCode = 00000002
eax=86399708 ebx=8180c584 ecx=8c1232d0 edx=8c123310 esi=00000000 edi=00000000
eip=8187ec58 esp=90706b80 ebp=90706b88 iopl=0        nv up ei pl nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000            efl=00010202
nt!KiReadyThread+0x3c:
8187ec58 8906            mov     dword ptr [esi],eax  ds:0023:00000000=????????
Resetting default scope
STACK_TEXT:
[...]
90706b88 8188080e 819cfc20 863998ac 863998b4 nt!KiReadyThread+0x3c
90706ba4 818808d2 00000001 00000000 00000000 nt!KiUnwaitThread+0x14a
90706bc0 8187a307 00000001 8c1d0d78 863998ac nt!KiWaitTest+0xb6
90706bd8 81882cff 863998ac 00000001 00000001 nt!KeReleaseSemaphore+0x4f
90706c04 81d8d741 8c1d0f8c 00000001 00000000 nt!AlpcpSignalAndWait+0x7f
90706c40 81db91dc 00000001 90706cac 00000000 nt!AlpcpReceiveSynchronousReply+0x33
90706cd0 81dc041c 8c172818 00020000 00ddfab0 nt!AlpcpProcessSynchronousRequest+0x648
[...]
```

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- Exploitation more difficult
  - Only candidate is smss.exe (despite *System*)
  - Unknown `PointerCount`
  - Requires advanced kernel pool feng-shui
    - `EPROCESS` takes 0x25c (604) bytes of NonPagedPool
    - failed attempt = Blue Screen of Death
- Definitely still possible!
  - keep an eye on my blog ☺

win32k!NtUserCheckAccessForIntegrityLevel use-after-free

- Impact
  - Local privilege escalation if exploitation succeeds
  - Denial of Service otherwise.
  - Windows Vista / 2008 Server only.
- Metrics
  - Memory: irrelevant
  - CPU time: irrelevant (instant)
- Fix
  - Setting the output object pointer to NULL in win32k!LockProcessByClientId
    - second dereference doesn't occur anymore

win32k!NtUserAttachThreadInput use-after-free

- Some threads in Windows are marked as GUI
  - can then talk to win32k.sys
  - required for anything graphics-related
- Every such thread has a kernel-mode message queue.

```
kd> dt tagQ
win32k!tagQ
   +0x000 mlInput                : tagMLIST
  [...]
   +0x13c QF_flags               : Uint4B
   +0x140 cThreads               : Uint2B
   +0x142 cLockCount             : Uint2B
   +0x144 msgJournal             : Uint4B
```

looks like refcounts!

win32k!NtUserAttachThreadInput use-after-free

- Threads can attach to each others' queues!
  - see AttachThreadInput (documented API)
- Queues must store # of reliant threads
  - uses `cThreads` for just that
- Queues freed in `win32k!UserDeleteW32Thread` when `(cThreads == 0) && (cLockCount == 0)`

```
.text:BF8D6B63 cmp     [ecx+tagQ.cLockCount], di
.text:BF8D6B6A jnz     short loc_BF8D6B7D
.text:BF8D6B6C mov     eax, ecx
.text:BF8D6B6E cmp     [eax+tagQ.cThreads], di
.text:BF8D6B75 jnz     short loc_BF8D6B7D
.text:BF8D6B77 push    eax                  ; Entry
.text:BF8D6B78 call    _FreeQueue@4         ; FreeQueue(x)
```

win32k!NtUserAttachThreadInput use-after-free

- There's no refcount leak in the implementation
  - no "free" incrementations
- Can we legitimately attach > 65,535 threads to a single queue?
  - Yes, if we can create that much.
  - Can we?
- Mark Russinovich had an excellent post about it, see *"Pushing the Limits of Windows: Processes and Threads"*

win32k!NtUserAttachThreadInput use-after-free

- Short answer: no on 32-bit Windows
  - limitations: kernel virtual address space size, physical memory capacity, …
  - only up to 32K threads, usually far less.
- Good news: yes on 64-bit Windows

win32k!NtUserAttachThreadInput use-after-free

Let’s test!

```
for (unsigned int i = 0; ; i++) {
   if (!CreateThread(NULL, 0,
                        (LPTHREAD_START_ROUTINE)ThreadRoutine,
                         NULL, 0, NULL)) {
        break;
   }
   printf(„threads: %u\n”, i);
}
```

win32k!NtUserAttachThreadInput use-after-free

```
c:\code\testlimit\objchk_win7_amd64\amd64>test
threads: 157179
c:\code\testlimit\objchk_win7_amd64\amd64>
```

- Windows 7 64-bit, 12GB of RAM
- ~ 2.64 GB physical memory consumption for 65,536 threads
- Several seconds of CPU time

win32k!NtUserAttachThreadInput use-after-free

Security by poor programming practices?

- Overflowing a 16-bit counter shouldn't take too long, right?
  - in theory...
- Every *"attach thread A to B"* request:
  - results in a full recalc of thread queues
  - takes `O(n²)` time, `n` = session thread count
- Creating a queue with 2^{16} threads takes ~2^{48} steps
  - could've been done a whole lot faster

win32k!NtUserAttachThreadInput
use-after-free

`AttachThreadInput(x,y)` algorithm (pseudo-code)

```
win32k!gpai.append(pair(thread_from, thread_to));
foreach thread in current_thread->desktop:
    pqAttach = thread->pq;
         changed = false;
         if thread->attached:
          continue
     do:
          foreach thread_nested in current_thread->desktop:
               if thread_nested->pq == pqAttach:
                    foreach req in win32k!gpai:
                         if req.first == thread_nested || req.second == thread_nested:
                              attach(req.first, req.second)
                              changed = true
     while changed;
```

win32k!NtUserAttachThreadInput use-after-free

- Still exploitable (with some extra work)
  - Note: recalc only for caller thread's desktop

- Plan:
  - Create `self_desktop` and `thread_desktop` desktops
  - Assign main thread to `self_desktop`
  - Create 65,536 threads
    - assign all to `thread_desktop`
  - Attach threads 1..65,536 to 0
    - fills in the win32k!gpai list with thread pairs
    - fast: single attach is O(1) for foreign desktops (no recalc)

[...]

win32k!NtUserAttachThreadInput use-after-free

- Plan, part two
  - Switch main thread and current workstation to `thread_desktop`
  - Attach main thread queue to thread 0
    - causes a full recalc, $n = 2^{16}$, $O(n^2) \sim 2^{32}$ iterations
      - within one syscall, no context switches
    - triggers the integer overflow; refcount = [...], 65536, 0
    - triggers a free of the shared input queue
  - Spray session paged pools
  - Terminate remaining threads
    - triggers use of the freed queue

win32k!NtUserAttachThreadInput use-after-free

Results

- Multiple assertion hits on a checked build

```
(s: 1 0x4dc.484 test.exe) [Err] DBGValidateQueueStates: Assertion failed: (pti == pq->ptiMouse)
    || (fAttached && (pq == pq->ptiMouse->pq))
(s: 1 0x4dc.484 test.exe) [Err] DBGValidateQueueStates: Assertion failed: (pti == pq-
    >ptiKeyboard) || (fAttached && (pq == pq->ptiKeyboard->pq))
```

- Ultimately, a bugcheck

```
win32k!DestroyThreadsMessages+0x22:
fffff960`0011a6b6 488b33   mov     rsi,qword ptr [rbx] ds:002b:aaaaaaaa`aaaaaaaa=????????????????
Resetting default scope
STACK_TEXT:
fffff880`fd18e7d0 fffff960`00119da9 : [...] : win32k!DestroyThreadsMessages+0x22
fffff880`fd18e800 fffff960`0013deb7 : [...] : win32k!xxxDestroyThreadInfo+0x1001
fffff880`fd18e8d0 fffff960`00115140 : [...] : win32k!UserThreadCallout+0x93
fffff880`fd18e900 fffff800`0299d375 : [...] : win32k!W32pThreadCallout+0x78
```

win32k!NtUserAttachThreadInput use-after-free

- Impact
  - Invincible processes by *infinite* loops in win32k.sys
  - Denial of Service (failed use-after-free exploitation)
  - Escalation of Privileges (successful exploitation)
    - resource constraints
    - kernel pool feng-shui required again
- Metrics
  - Memory: ~2.5GB required for thread storage.
  - CPU time: up to 10 minutes
    - creating threads (2^{16} steps): < 5s
    - attaching threads (2^{16} steps) < 2 minutes
    - doing global recalc (2^{32} steps) < 10 minutes

win32k!NtUserAttachThreadInput use-after-free

The fix

- Expand the `cThreads` / `cLockCount` refcounts to 32 bits
  - you can't possibly have 4,294,967,296 threads... yet (but ping me when you can)

```
[...]
+0x140 cThreads        : Uint2B
+0x142 cLockCount      : Uint2B
[...]
```

→

```
[...]
+0x140 cThreads        : Uint4B
+0x144 cLockCount      : Uint4B
[...]
```

```
[...]
add     word ptr [r12+140h], 1
[...]
```

→

```
[...]
inc    dword ptr [r12+140h]
[...]
```

win32k!NtGdiAddFontResource use-after-free

- Applications can load external fonts for local usage
  - documented AddFontResource Windows API
  - perhaps used in every win32k.sys font fuzzer

" When an application no longer needs a font resource it loaded by calling the AddFontResourceEx function, it must remove the resource by calling the RemoveFontResourceEx function.
"

- Sounds reference-countable! :-)

win32k!NtGdiAddFontResource use-after-free

- Indeed...

Callstack

```
kd> kb
ChildEBP RetAddr  Args to Child
9b714af4 [...] win32k!PFFOBJ::vLoadIncr+0x12
9b714b14 [...] win32k!PFTOBJ::chpfeIncrPFF+0x94
9b714b80 [...] win32k!PUBLIC_PFTOBJ::bLoadFonts+0x90
9b714bc8 [...] win32k!GreAddFontResourceWInternal+0xad
9b714d14 [...] win32k!NtGdiAddFontResourceW+0x15e
9b714d14 [...] nt!KiFastCallEntry+0x12a
0022fd2c [...] ntdll!KiFastSystemCallRet
```

win32k!NtGdiAddFontResource use-after-free

- Indeed...

```
.text:BF8149BF ; public: void __thiscall PFFOBJ::vLoadIncr(unsigned long)
[...]
.text:BF8149C4                 test    [ebp+arg_0], 20h
.text:BF8149C8                 mov     eax, [ecx]
.text:BF8149CA                 jz      short loc_BF8149D1
.text:BF8149CC                 inc     dword ptr [eax+28h]
.text:BF8149CF                 jmp     short loc_BF8149D4
.text:BF8149D1
.text:BF8149D1 loc_BF8149D1:
.text:BF8149D1                 inc     dword ptr [eax+24h]
.text:BF8149D4
.text:BF8149D4 loc_BF8149D4:
.text:BF8149D4                 call    PFFOBJ::vRevive(void)
.text:BF8149D9                 pop     ebp
.text:BF8149DA                 retn    4
.text:BF8149DA ?vLoadIncr@PFFOBJ@@QAEXK@Z endp
```

refcount incrementation!

win32k!NtGdiAddFontResource use-after-free

- Details
  - 32-bit refcount involved on both X86 / X86-64
    - perhaps an `ULONG`, but exact structure unknown
  - No persistent memory allocations!
- How long does it take?
  - well, 2^{32} system calls...
  - test environment: Windows XP SP3 in a VM, single core
  - incr. rate at about 100,000 requests / second
  - (only) ~12 hours!
    - could be less on better machine or with optimized exploit

win32k!NtGdiAddFontResource use-after-free

- Results
  - Upon unload, the `PFFOBJ` class is "killed" when refcount drops to 0.
  - Stack trace:

```
#0 win32k!PFFOBJ::vKill
#1 win32k!PFFOBJ::bDeleteLoadRef
#2 win32k!PFTOBJ::bUnloadWorkhorse
#3 win32k!GreRemoveFontResourceW
#4 win32k!NtGdiRemoveFontResourceW
```

win32k!NtGdiAddFontResource use-after-free

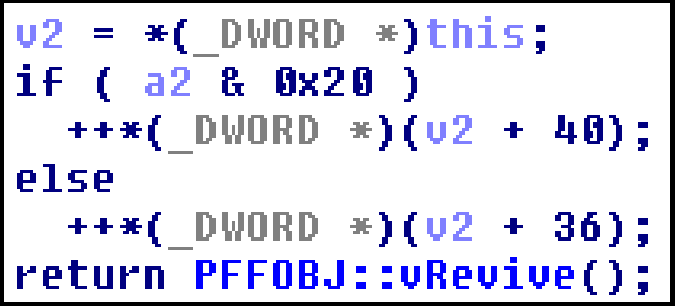
- All sorts of badness
  - use-after-frees
  - NULL pointer dereferences

```
kd> g
Access violation - code c0000005 (!!! second chance !!!)
win32k!bGetNtoD_Win31+0x1f:
82008864 8b4830          mov     ecx,dword ptr [eax+30h]
kd> ? eax
Evaluate expression: 0 = 00000000
kd> kb
ChildEBP RetAddr  Args to Child
9bb28bc4 [...] win32k!bGetNtoD_Win31+0x1f
9bb28bf8 [...] win32k!PFEOBJ::bSetFontXform+0x3e
9bb28c98 [...] win32k!RFONTOBJ::bInit+0x1bf
9bb28cb0 [...] win32k!RFONTOBJ::vInit+0x16
9bb28cd4 [...] win32k!GreGetRealizationInfo+0x2a
9bb28d24 [...] win32k!NtGdiGetRealizationInfo+0x41
9bb28d24 [...] nt!KiFastCallEntry+0x12a
```

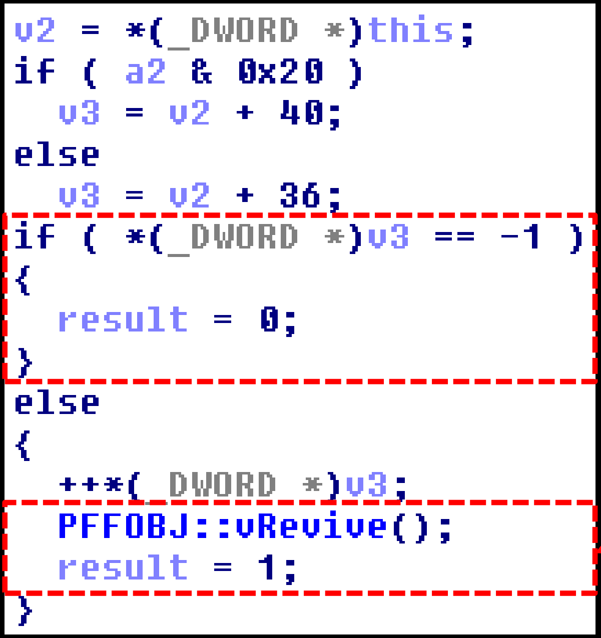
win32k!NtGdiAddFontResource use-after-free

- Impact
  - typically DoS or EoP, depending on exploitation skills
  - works on 32-bit and 64-bit platforms
- Fix: mount a reference count limit at `ULONG_MAX`
  - Quite risky, what if there's a two-thread race?

win32k!NtGdiAddFontResource use-after-free



```
v2 = *(_DWORD *)this;
if ( a2 & 0x20 )
  ++*(_DWORD *)(v2 + 40);
else
  ++*(_DWORD *)(v2 + 36);
return PFFOBJ::vRevive();
```



```
v2 = *(_DWORD *)this;
if ( a2 & 0x20 )
  v3 = v2 + 40;
else
  v3 = v2 + 36;
if ( *(_DWORD *)v3 == -1 )
{
  result = 0;
}
else
{
  ++*(_DWORD *)v3;
  PFFOBJ::vRevive();
  result = 1;
}
```

overflow prevention check

hmmm... a new bug?

Also worth checking out

CVE-2011-2013 (tcp/ip stack use-after-free)

- Fixed on November 8, 2011
- 32-bit reference counter integer overflow
- Remote, through UDP packets!
- Works on closed ports!

- Root cause - adverse circumstances and no mitigations
  - "small" integer, a 64-bit one would suffice
  - no sanity checks
  - no persistent memory allocations bound to refcount incrementations

Mitigations concepts

Preventing refcount problems

- You can't prevent developers from writing buggy code
- But you can mitigate consequences of the resulting vulns
    - Provide a *"secure"* interface for everyone to use
    - Not perfect, but raises the bar

Preventing integer overflows

- Introduce `refcount_t` as an alias to `int64_t`
    - doesn't cost anything: memory is cheap
        - times when it mattered are long gone
    - would prevent 99% refcount overflow attacks
    - potential problem: sometimes counters are difficult to recognize

Preventing integer overflows

- Introduce generic APIs for refcount manipulation
  - nt!IncrementRefcount, nt!DecrementRefcount, nt!TestRefcount
  - could include basic sanity checks

```
if (++(*refcount) < 1) {
  KeBugCheckEx(REFCOUNT_GONE_WRONG);
}

if (--(*refcount) < 0) {
  KeBugCheckEx(REFCOUNT_GONE_WRONG);
}
```

Preventing excessive dereferences

- Much harder than plain integer problems
    - let's never free refcounted allocations! :-)
        - revisit the idea when unlimited memory available
        - curio: *nt!NtMakePermanentObject*
            - requires *SeCreatePermanentPrivilege*

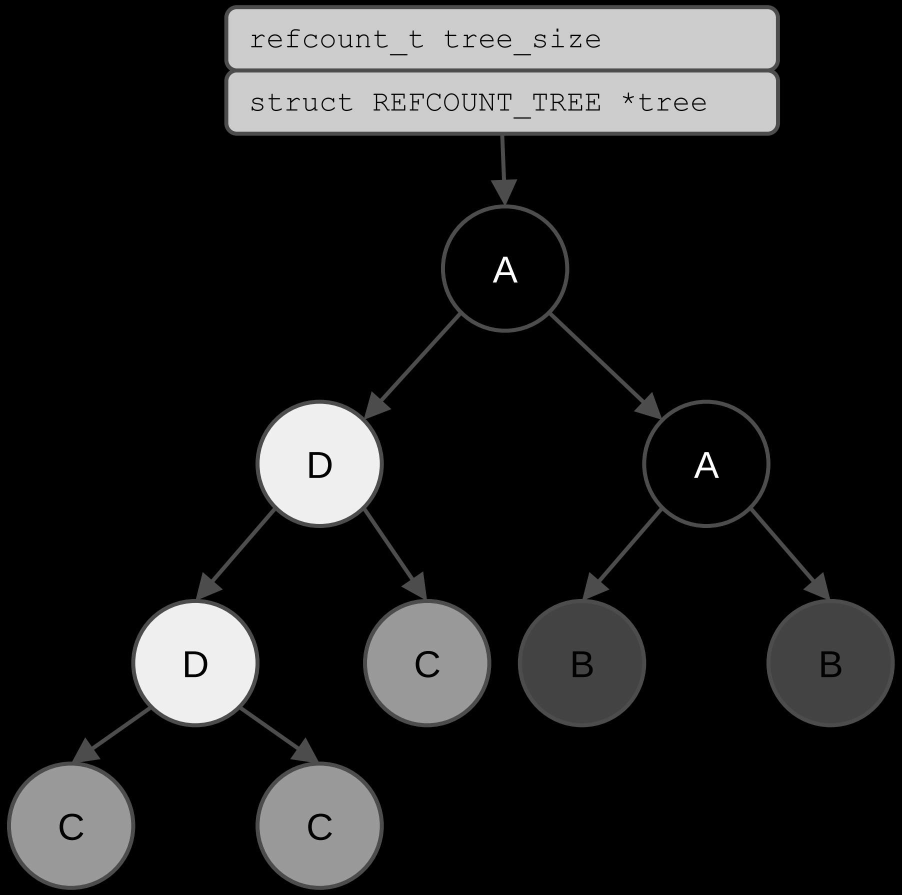
- The interface doesn't know caller's logic
    - which derefs are paired with which refs?

Preventing excessive dereferences

- Idea: identify each ({reference}, {dereference}) pair with a unique tag
  - similarly to *pool tags*
- A "reference counter" becomes a "reference tree"
- Store information about all pending reference tags in the tree
- Always pass the tag to the {ref,deref} API
  - Test if tag is in tree before decrementing

Preventing excessive dereferences

- A self-balancing binary search tree



AVL trees already implemented in Windows

Preventing excessive dereferences

Performance hit

	Tree implementation	Traditional implementation
Reference cost	O(lg n)	O(1)
Deference cost	O(lg n)	O(1)
Test cost	O(1)	O(1)

- Statistics (Windows 7 SP1 32-bit with a few apps)
  - Average *PointerCount*: 118009 / 29364 =~ 4.01883
  - Average *HandleCount*: 15135 / 29364 =~ 0.51542
- Difficult to measure refs/derefs per second
- Overhead should be acceptable (own opinion)

Preventing excessive dereferences

Memory overhead

- Loose estimate
  - ~120,000 references to NT executive objects at startup
  - Twice as much during typical session =~ 250,000
  - Twice as much including other refcounts =~ 500,000
  - Assume 64 bytes per one reference
    - pool header, tag, pointers to parent / children
  - A total of extra ~30MB of NonPaged memory
    - guess if my 12GB RAM machine can take it?

Preventing excessive dereferences

Other problems

- Lazy developers
  - would have to define unique tags
  - already do it for pool allocations, so perhaps possible?
- Legacy issues
  - existing API routines lack tagging information
    - `ObReferenceObject{ByHandle,ByPointer}`
  - how to communicate failure (e.g. lack of memory)?
- Passing tags through wrappers
- Possibly low engineering effort / benefit ratio
  - how many bugs would this prevent?

Preventing excessive dereferences

Benefits

- If properly executed, would prevent most use-after-frees through double derefs
  - *stealing* references not possible anymore
  - *dereference* sequence would have to match the *reference* one to exploit

- Automatic mitigation integer overflow
  - through memory constraining

- Robust interface for future use

Conclusions

Random thoughts

- Refcounts bugs = use-after-frees
  - otherwise rarely observed (perhaps except Tarjei)
  - usually time-consuming and tricky to exploit
  - often memory-consuming
- Kernel pool spraying should be better investigated
- Integer types != machine word don't scale
  - No explicit (1/2 void*) or (1/4 void*)
  - Small types used 20 years ago can take revenge
  - More to be found?

Random thoughts

- Inconsistent patches
  - sometimes extending types
  - sometimes pinning
  - sometimes sanity checks
  - would a common interface help?
- Microsoft doesn't backport fixes?
  - Why CVE-2010-2549 only affected Vista / 2008?
  - Could've been found by bindiffing?
  - See Nikita's talk

Благодарю вас за внимание!

Questions?

E-mail: j00ru.vx@gmail.com
Blog: http://j00ru.vexillium.org/
Twitter: @j00ru